人工智能实验

报 告

|  |  |
| --- | --- |
| **姓名** | 张子洋 |
| **学号** | 8208221223 |
| **班级** | 2203班 |
| **专业** | 计算机科学与技术 |
| **年级** | 22级 |
| **时间** | 2024.3 |

中南大学计算机学院

2024

## 实验内容

黑白棋(Reversi)，也叫苹果棋，翻转棋，是一个经典的策略性游戏。一般棋子双面为黑白两色，故称“黑白棋”。因为行棋之时将对方棋子翻转，变为己方棋子，故又称“翻转棋”(Reversi)。棋子双面为红、绿色的称为“苹果棋”。它使用8x8的棋盘,由两人执黑子和白子轮流下棋，最后子多方为胜方。随着网络的普及，黑白棋作为一种最适合在网上玩的棋类游戏正在逐渐流行起来。中国主要的黑白棋游戏站点有Yahoo游戏、中国游戏网、联众游戏等。

游戏规则：

棋局开始时黑棋位于e4和d5，白棋位于d4和e5。

1. 黑方先行，双方交替下棋。

2) 一步合法的棋步包括：在一个空格新落下一个棋子，并且翻转对手一个或多个棋子。

3) 落下的棋子与棋盘上已有的同色棋子间，对方被夹住的所有棋子都要翻转过来。可以是横着夹，竖着夹，或是斜着夹。夹住的位置上必须全部是对手的棋子，不能有空格。

4) 一步棋可以在数个（横向，纵向，对角线）方向上翻棋，任何被夹住的棋子都必须被翻转过来，棋手无权选择不去翻某个棋子。

5) 除非至少翻转了对手的一个棋子，否则就不能落子。如果一方没有合法棋步，也就是说不管他下到哪里，都不能至少翻转对手的一个棋子，那他这一轮只能弃权，而由他的对手继续落子直到他有合法棋步可下。

6) 如果一方至少有一步合法棋步可下，他就必须落子，不得弃权。

7) 棋局持续下去，直到棋盘填满或者双方都无合法棋步可下。

8) 如果某一方落子时间超过1分钟，则判该方失败。

**实验要求**

1. 使用MCTS算法实现Mini AlphaGo for Reversi, 有能力的同学可以采用最大最小算法进行对比.
2. MCTS算法部分需要自己实现，尽量不使用现成的包，工具或者接口.实验报告不能直接复制、粘贴网上内容，请理清思路，转化为自身知识.珍惜每一次训练机会.
3. 在博弈过程中，Mini AlphaGo每一步所花费的时间以及总时间需要显示出来.
4. 需要有简单的图形界面用于人机博弈交互.
5. 可使用Python语言，也可以使用其他编程语言实现.

## 实验方案

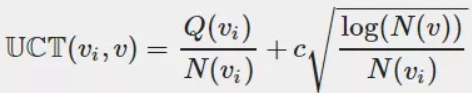
首先是介绍**蒙特卡洛算法**（MCTS），有四步：

**1.选择（Selection）：**

从根结点R开始，选择连续的子结点向下至叶子结点L。下面的结点有更多选择子结点的方法，使游戏树向最优点扩展移动，这是蒙特卡洛树搜索的本质。

未访问：还没有评估过当前局面。  
未完全展开：被评估过至少一次，但是子节点（下一步的局面）没有被全部访问过，可以进一步扩展。  
完全展开：子节点被全部访问过。

我们通过下面这个公式来进行每个结点的优劣的比较。



**2.扩展 (expansion）：**

除非任意一方的输赢导致游戏结束，否则L会创建一个或多个子结点或从结点C中选择。

将刚刚选择的节点加上一个统计信息为「0/0」的节点，存储当前节点模拟的次数和赢的次数。

1. **模拟/仿真（Simluation）：**

我们每个节点（每个节点代表每个不同的局面）都有两个值，代表这个节点以及它的子节点模拟的次数和赢的次数，比如模拟了 10 次，赢了 4 盘，记为 4/10。

1. **回溯/反向传播（Backpropagation）：**

类似于递归里的回溯，就是从子节点开始，沿着刚刚向下的路径往回走，沿途更新各个父节点的统计信息。

最后我们继续循环这个过程，一直迭代，模拟出较合适的结果。

**Minimax算法**

Minimax算法大多被用于棋类游戏中，是一种找出失败的最大可能性中的最小值的算法，即最小化对手的最大得益的算法，就是说想将自己的分数尽量高，而对手尽量选取小的值的算法。

局面估价函数：我们给每个局面（state）规定一个估价函数值 f，评价它对于己方的有利程度。胜利的局面的估价函数值为 +oo，而失败的局面的估价函数值为–oo。

Max 局面：假设这个局面轮到己方走，有多种决策可以选择，其中每种决策都导致一种子局面（sub-state）。由于决策权在我们手中，当然是选择估价函数值 f 最大的子局面，因此该局面的估价函数值等于子局面 f 值的最大值，把这样的局面称为 max 局面。

Min 局面：假设这个局面轮到对方走，它也有多种决策可以选择，其中每种决策都导致一种子局面（sub-state）。但由于决策权在对方手中，在最坏的情况下，对方当然是选择估价函数值 f 最小的子局面，因此该局面的估价函数值等于子局面 f 值的最小值，把这样的局面称为 max 局面。

α-β剪枝：减少搜索次数，优化时间。

**比较：**

1.搜索空间：

MCTS：MCTS 通过随机模拟和统计学习，对可能的决策路径进行搜索，可以处理大型和复杂的搜索空间，特别适用于状态空间较大的游戏，如围棋。

最大最小算法：最大最小算法完全搜索游戏的博弈树，因此对于状态空间较小的游戏效果更好，如井字棋或五子棋。

2.搜索深度：

MCTS：MCTS 不需要事先指定搜索深度，它通过动态扩展搜索树的方式进行搜索，可以根据实际情况自适应地调整搜索深度。

最大最小算法：最大最小算法需要在每一步决策时指定搜索深度，搜索深度的增加会显着增加计算时间，因此对于深度受限的游戏或资源有限的环境，可能不太适用。

3.对抗性：

MCTS：MCTS 通过随机模拟和统计学习，可以处理复杂的对抗性情况，并在长期策略上表现良好。

最大最小算法：最大最小算法在每一步都选择最优的决策，但是它的效果受限于搜索深度和启发式函数，容易受到对手的干扰。

4.适用性：

MCTS：MCTS 适用于大型状态空间、复杂对抗性和随机性较高的游戏，如围棋、星际争霸等。

最大最小算法：最大最小算法适用于状态空间较小、对抗性较弱的游戏，如井字棋、五子棋等。

## **功能模块**

**MCTS文件：**

def getInitialBoard():初始化棋盘数组。

def countTile(board, tile):返回棋子数用于判断胜负。

def possible\_positions(board, tile):返回一个颜色棋子可能的下棋位置，用于判断分支。

def isOnBoard(x, y):判断是否在棋盘上。

def updateBoard(board, tile, i, j, checkonly=False):检查是否是合法走法，如果合法返回需要翻转的棋子列表。其中board代表游戏板，tile代表玩家的图块（1 或 2），并(i, j)代表玩家想要放置图块的位置。如果checkonly设置为True，该函数仅检查移动的有效性，而不实际更新棋盘。该函数将玩家的棋子暂时放置在指定位置，并在所有八个方向检查对手的棋子是否可以翻转。如果对手的牌可以翻转，则会将其位置添加到need\_turn列表中。检查完所有方向后，该函数会从棋盘上移除临时放置的棋子。如果checkonly是False，则根据游戏规则，通过放置玩家的牌并翻转对手的牌来更新棋盘。最后，它返回对手被翻转的牌数。

def mctsNextPosition(board):

UCB1 计算：该函数ucb1计算树中给定节点的UCB1值。它考虑了节点被访问的次数 (nplayout)、在节点获得的奖励以及模拟总数 。

模拟：采用递归函数，从给定的棋盘状态开始执行播出模拟。它随机选择合法的动作，直到游戏达到最终状态（赢、输或平局）。它评估最终的棋盘状态以确定比赛的结果。

扩展：该函数expand通过考虑当前玩家所有可能的合法动作来生成给定棋盘状态的子节点。

树遍历：该函数find\_path从根节点开始执行树遍历。它根据 UCB1 值选择子节点，直到到达叶节点。

回溯：在主循环内，算法执行一系列模拟（由变量控制loop）来构建蒙特卡罗树。对于每次模拟，它都会选择树中的一条路径，执行模拟，并更新路径沿线节点的统计数据（访问计数和奖励）。

选择最佳走法：模拟完成后，该函数选择在比赛期间获得的平均奖励最高的走法，并将其返回为计算机玩家的下一步走法。

for loop in range(0, 5000):

这是一个循环，它迭代了一定次数，这里迭代了 5000 次。

current\_path = find\_path(root, loop)：根据当前的搜索树和迭代次数，找到当前路径，即当前要走的一系列棋步。

tile = COMPUTER\_NUM：设定当前轮到的玩家，开始时是电脑。遍历当前路径中的每一步。根据当前玩家（电脑或玩家）的回合和路径中的棋步，更新棋盘。

增加当前节点的模拟对局次数。

如果当前对局获胜，则增加当前节点的奖励值。

if t\_playout >= 5 and len(t\_childrens) == 0:如果当前节点的模拟对局次数达到一定阈值，并且当前节点没有子节点（即还未进行扩展），则扩展当前节点的子节点。

然后将当前节点更新为其子节点，以便下一次迭代继续向下搜索。

这样，每次迭代都会更新搜索树，并根据模拟对局的结果调整节点的奖励值，从而逐步优化算法的决策过程。

**GAME文件：**

1.游戏板（ReversiBoard类）：

用代表板单元的矩形网格初始化我们的棋盘。self.bind("<1>", self.put\_stones) 将鼠标左键单击事件绑定到put\_stones函数中，实现在棋盘上放置棋子的方法。处理玩家的移动并在玩家移动后触发AI\_move方法，该方法使MCTS算法来实现AI的移动。每次移动后，它都会检查游戏的结束条件并显示包含结果的消息框。

在该put\_stones方法中，在触发 AI 移动之前，我检查自上次移动以来是否已经过去了一分钟以上。如果是，则游戏宣告结束，玩家输。refresh方法在每次移动后更新棋盘上棋子的显示。

2.主应用程序窗口（ReversiBoard类）：

\_\_init\_\_方法初始化框架。self.master.title("黑白棋")将窗口标题设置为“黑白棋”，ReversiBoard被实例化为self.f\_board，代表游戏板。将游戏板小部件打包到框架中，并在其周围填充一些内容。

3.结果消息框：

游戏结束后，将显示一个消息框，显示结果（获胜/失败/平局）。

4.总时间测量：

在AI移动过程中，每次移动所花费的时间都会打印在控制台中。游戏结束后，计算并打印 AI 移动所花费的总时间。在玩家操作之前会进行判断，如果时间超过60s那么此时再落子就会弹出失败窗口。

## 总结

时间比较紧张，需要学习MCTS，要完成的功能较多，python还是比较生疏。系统的代码仍有一些漏洞。并且修改了源代码的一些漏洞，可以判断玩家操作是否超时，修复在游戏结束后还可以在棋盘上落黑子来继续已结束的棋盘的bug，我添加了sys.exit()函数，使得可以在游戏结束后直接退出程序。

写完实验后我对MCTS算法有了更深的理解，对python代码的操作和对bug的修改也有了进步。